4

Final Report

Sponsored by

Defense Advanced Research Projects Agency (DOD)
Information Processing Technology Office

Building Software Systems Economically with Mechanized Logic: Initial Design Proposal

ARPA Order No. 5246 ✓

Issued by Naval Electronic Systems Command under
Contract N00039-85-K-0085 ✓

Donald I. Good
J Strother Moore
Matt Kaufmann

June 30, 1987

Institute for Computing Science
2100 Main Building
The University of Texas at Austin
Austin, Texas 78712
(512) 471-1901

Accession For		
NTIS GRA&I		☒
DTIC TAB		☐
Unannounced		☐
Justification		
By		
Distribution/		
Availability Codes		
Dist	Avail and/or Special	
A-1		

87 5513

87 12 3 026

This report summarizes the work done under this contract in the context of the long-term research plan described in [14]. That paper, which was formulated under the auspices of this contract, outlines a plan for the development of Rose, an applicative language based on a formal logic with powerful mechanical proof assistance. We report here the progress to date on Rose, including our related efforts, following fairly closely the outline of the Research Plan [14]. The first two sections of the Research Plan, **Introduction and Historical Foundations**, provide additional background on our perspective; we omit them from our outline here, however. Instead, we include a brief summary of our study of related work in the first section below.

We should mention that Rose will grow out of the existing computational logic of Boyer and Moore, described in [6, 7, 8]. Indeed, we identify the current version of the Rose language and logic with the current Boyer-Moore computational logic.

1 Study of Related Work

During the second quarter of 1985, we participated in a close up evaluation of three other major verification systems (along with our own Gypsy system): GE's Affirm, SDC's Ina Jo/FDM, and SRI's Revised Special/HDM. A week long visit was in fact made to each of these sites to study and use the local verification system and to discuss future verification directions with the local developers. The results of these visits are described in a sequence of Internal Notes [11, 12, 13]. The entire effort's conclusions appear in the The Kemmerer study report [19].

2 Mechanizing Rose Logic

Our goal is to develop an economical technology for building proved computing systems with mechanized formal logic. The unifying element of this technology is the functional language Rose which we are designing. Rose embodies a powerful formal logic, and it also is an executable, functional programming language. Thus, potentially, Rose provides a single, unified formalism that can express both hardware and software systems and their specifications and requirements.

In the long term, with the development of parallel architectures and optimizing compilers that exploit theorem proving, we believe that functional programming languages will be useful across a wide variety of tasks. In the intermediate term, we intend that Rose be convenient for software applications such as encryption boxes, flow modulators, message servers, etc. These are the applications areas in which Gypsy commonly is used today. In the short term, we intend that Rose be a convenient language in which to specify and prove properties about von Neumann computing systems.

The purpose of this phase of our work is to mechanize the Rose logic so that it can be used extensively and economically in all of the previous kinds of activities. We will do this by increasing the power of current Boyer-Moore logic and its theorem prover, by defining the Rose language which embodies the expanded logic and presents it in a more conventional and familiar notation, and by implementing a life-cycle support system for Rose that supports the development and maintenance of large collections of Rose functions, theorems, and proofs.

2.1 Rose Logic

Rose logic will ultimately be current Boyer-Moore logic extended to include

1. quantification over finite domains,
2. a simulation of functions as first-class objects,
3. partial recursive functions.

Much research has already been carried out by Boyer and Moore [8] to support these modifications.

An experimental version of the theorem prover supporting quantification over finite domains and partial functions exists, and it is being tested. The steps necessary to release it for wide-spread use are:

1. convince ourselves and our peers that the modified logic is consistent,
2. convince ourselves and our peers that the modifications made to the released version of the theorem prover are correct, and
3. write the manual for the new logic and theorem prover.

To these ends, a report on the extended Boyer-Moore logic and theorem prover has been completed [8]. A draft of a detailed user's manual has also been completed [5], describing not only the basics of using the theorem prover but also containing many helpful tips for using it efficiently. It also serves the role of being a reference guide for the logic as it currently exists.

2.2 Rose Language

As mentioned above, the current Rose logic is the existing Boyer-Moore logic. What we desire is, at the least, a more conventional and familiar notation for Rose logic than the Lisp notation that presently is used in Boyer-Moore logic.

But the Rose language will evolve from the current Boyer-Moore logic in other ways besides sugaring the syntax. For example, we expect Rose to contain mutual recursion and (more generally) a relaxation on the current Boyer-Moore restrictions on the order of definitions. We also anticipate the inclusion of name space control (scopes), a simulation of functions as first-class objects, type-checking, and iterative forms and partial functions such as those already existing in the experimental new version of the Boyer-Moore logic and prover [8].

In order to aid the development of the Rose language, a formal semantic definition of the language Micro Gypsy (discussed below) was developed in an experimental Rose syntax [15]. This definition is the basis for proving the correctness of the Micro Gypsy compiler. In addition, the type mechanism in the Rose language was investigated by considering the difficulty of expressing, in Rose, the algorithms for checking the well-formedness of Micro Gypsy expressions [22].

2.3 Rose Support System

An experimental window-based interface to the Boyer-Moore prover was developed for Symbolics Lisp Machines [2]. Although we expect to redesign this interface, its development provided valuable experience.

2.4 Document Management

Preliminary investigation was made into the design of a Rose Development System. This system would maintain consistency among related documents such as source, object, manuals, and so on. So far, the most promising approach to document management that we have discovered is the Neptune hypertext system [9] being developed by Tektronix to support CAD (Computer Aided Design) and CASE (Computer Aided Software Engineering) systems. More thoughts on this matter may be found in the Research Plan [14].

2.5 Theory Management, Reusable Theories

Some thought has been given to implementing a hierarchical library structure that allows one to merge theories. This turns out to be a somewhat complicated issue in the setting of the current Boyer-Moore system, but we believe such an improvement to be feasible. We have found it quite helpful to reuse theories -- for example, we have libraries of arithmetic facts and facts about subsets that have been used more than once -- and a hierarchical library structure would encourage more theory reuse.

2.6 A "Smart" Blackboard

We imagine the user developing a system and its proof in a medium as flexible as a blackboard but which, unlike a blackboard, is active and is capable of manipulating the formulas inscribed on it as well as following the arguments about them. We already mentioned the White Rose interface above, which is an early step in this direction. In addition, an interpreter has been developed which includes a trace and break package as well as a user's guide and technical documentation [1, 3]. (We are well aware that executability is extremely important in the development/acquisition of specifications.) Another feature of this electronic blackboard should be a convenient means for querying the Boyer-Moore database. Some recent additions made to the system for this purpose are documented in Chapter 12 of [5].

2.7 Building Trusted Systems

The mechanized logic whose development is described above will be used in building a variety of trusted systems. As the power of the Rose system evolves, proofs of both von Neumann and functional computing systems will be constructed. Conversely, use of the system will provide important feedback into the development of the Rose logic, language, and support system.

The applications of Rose that we foresee include the following:

- a formal definition of the Rose language,
- a formal definition for a subset of Ada,
- a formal definition of the Micro Gypsy language,
- a formal definition of the FM8501 assembly language,
- a formal definition of FM8501' (a successor to FM8501)
- a proof of correctness of a Micro Gypsy compiler to FM8501 (and FM8501'),
- a proof of correctness of a Micro Gypsy run-time executive for FM8501',
- a proof of correctness of an FM8501 (and FM8501') assembler,
- a proof of correctness of a Rose compiler,
- a proof of correctness of a Rose proof checker.

The remaining sections below report our progress toward proving correctness of von Neumann systems and functional systems, respectively.

3 Proving von Neumann Systems

Work proceeded toward the goal of producing a *vertically verified system*, i.e. a system which has been proved correct from the high-level language through the operating system and down to the hardware level. The paper [4] describes this work in some detail. There are three components to our vertically verified system: the machine (including the hardware and assembler), the operating system, and the systems programming language (including a compiler and a parser). The hardware, operating system, and compiler are independent doctoral dissertation research projects; the latter two of these are works in progress. We discuss these all in turn below, excepting the assembler (which is work in progress under other support). Once the three components are completed, their integration into a single system can proceed.

Figure 1 is taken from the paper [4], and illustrates our plan to achieve vertical verification. A quite thorough explanation of this fundamental diagram may be found in [4]; here is a summary. Consider for example the bottom parallelogram of this figure. There is a notion of an *abstract* FM8501 state, i.e. a state as seen at the level of the machine instruction set. There is also the notion of a *concrete* FM8501 state, i.e. a state as seen at the level of state-holding devices and combinational logic; this consists of an abstract state (a *programmer-visible* state) together with an *internal* state. Now suppose that one starts with an abstract state, as represented by the FM8501 box on the left side of the figure. The downward arrow from that box represents the result of "completing" this abstract state to an appropriate concrete state. The left-to-right arrow from the FM8501 box represents the "abstract run" of a given number of instruction steps on that state, while the arrow below it represents the "concrete run" of a corresponding (larger) number of instructions on the corresponding concrete state. The upward arrow on the lower right completes the diagram, which means roughly that if one takes the concrete state resulting from the "concrete run" and abstracts from it a corresponding abstract state, then the result is the abstract state resulting from the "abstract run".

3.1 Hardware

We have designed and proved a microprocessor, called the FM8501, a conventional von Neumann engine of roughly the complexity of a PDP-11.

FM8501 is a complete, stand-alone microprocessor with a symmetrically organized instruction set. Its features include:

- 16-bit general purpose processor
- word addressing yielding a 64K word (128K byte) memory size
- eight general purpose registers (one also being the program counter)
- 16-bit instructions
- register-register, register-memory, or memory-memory operation is allowed with all instructions
- two-address instruction format
- register, register indirect, register indirect with post-increment, or register indirect with pre-decrement addressing mode are individually supported for both operands for all instructions
- general-purpose conditional move instruction
- Boolean, natural number, and integer operational specification
- separate ALU for effective address generation
- memory mapped I/O
- compact functional description

FM8501 is a micro-coded device. The microcode is used to control instruction decoding and internal data movement. A separate ALU is used for effective address calculations, increasing the performance of the microprocessor.

All registers may be used as index registers or as software stack pointers. Four status bits -- carry (C), overflow (V), negative (N), and zero (Z) -- can be conditionally set by every instruction. FM8501 can access 2^16 memory locations, each one word (16-bits) in size; FM8501 can directly manipulate 128K Bytes of memory.

All FM8501 instructions are one word (16-bits) in size. Every instruction specifies a source and a destination location, each of which is either in a register or in memory. Instructions for the FM8501 specify two kinds of information: the operation to be performed and the location of the operands on which the operation is performed. Every instruction has a source and a destination. If two sources are required the destination operand serves as the other source before being modified (i.e., FM8501 has a two-address architecture). Because there are no special instructions for I/O, input/output devices are connected to FM8501 as memory devices (memory-mapped I/O).

We have proved the FM8501 in the following sense. The specification of the machine is an instruction interpreter for its machine language. The interpreter is defined as a self-recursive function with each

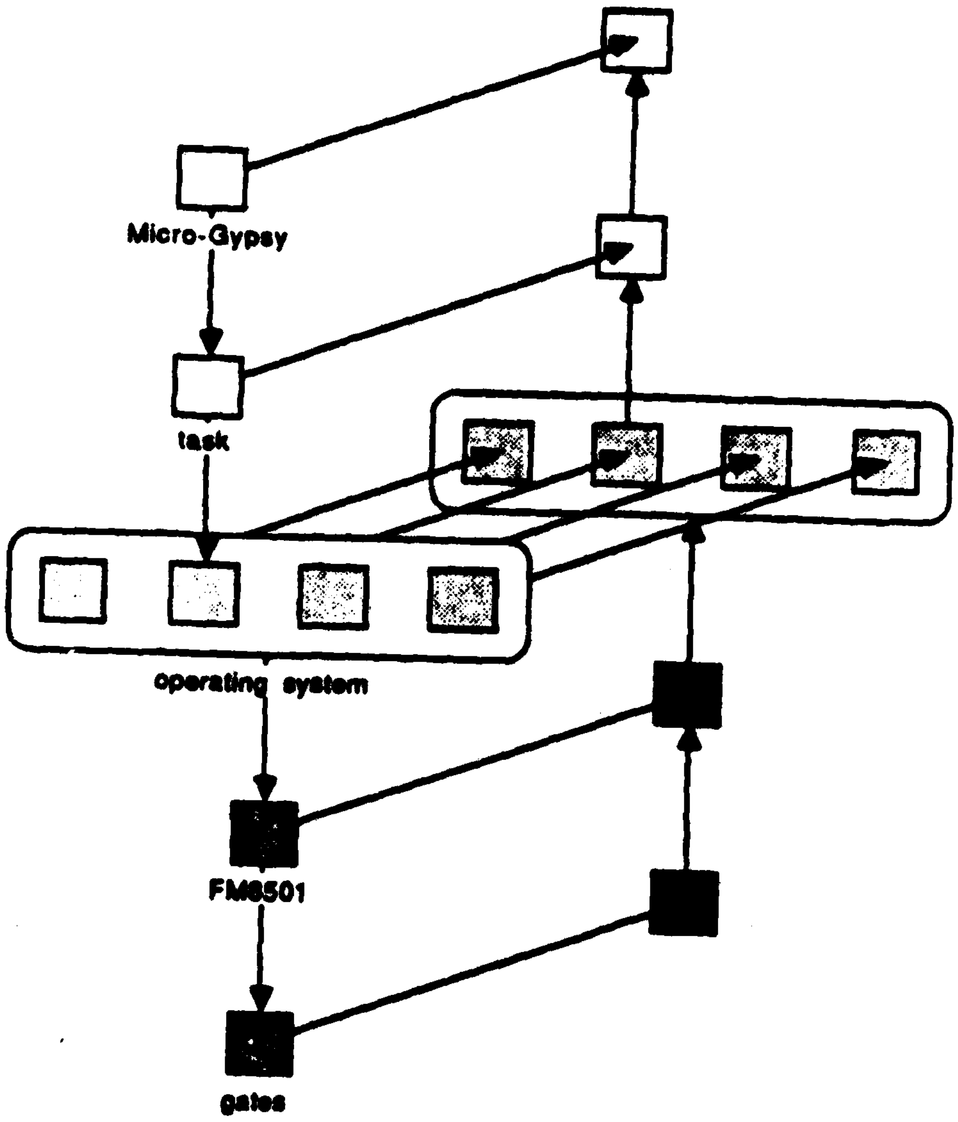

Figure 1: A Vertically Verified System

recursion corresponding to a single state transition. This interpreter formally specifies the effect of executing each possible instruction and may be thought of as a formal version of a programmer's manual for the device. The implementation of the FM8501 is a gate graph containing about 1700 Boolean gates, not counting those necessary to implement registers, latches, memory, etc. We have mechanically proved that the gate graph logically implements the instruction interpreter.

3.2 The Separation Kernel

Implementing the Rose runtime support software in Micro Gypsy requires multi-tasking. We are working on the proof of correctness of a small multi-tasking operating system designed for a simple von Neumann computer. The verification of the operating system includes two kinds of properties:

- Task isolation. We prove that the operating system, running on a single hardware processor, simulates a fixed number of isolated parallel tasks.
- Correctness of operating system services. The operating system provides the following services not provided by the bare target machine: message passing among tasks, and character I/O primitives to asynchronous devices.

The statement of the problem requires the definition of three machines: a task, an abstract operating system, and the target machine on which the operating system will run.

A task is modeled as a single address space of the target machine, plus the shared resources necessary to implement communication with other tasks and devices. This model ensures that a task's address space is isolated in the sense that no other task can perform a transition on it.

The abstract operating system specifies an operating system which manages a fixed number of tasks. The functionality specified for this operating system includes a round-robin scheduler, an error trap routine, I/O interrupt handlers, and supervisor service handlers for message passing and I/O.

The target machine is a two-state machine (supervisor and user modes) with I/O interrupts and with memory protection provided by base/limit registers. The instruction set and addressing modes are conventional, resembling a subset of the capabilities of a PDP-11. The operating system which is ultimately verified is written in the machine code of this target machine.

The correctness proof of the operating system takes two steps. First, we prove that the abstract operating system implements a system of parallel processes. This correctness theorem states that any task running under the abstract operating system behaves in a way identical to the model of an isolated task. Second, we prove that the target machine running the machine code version of the operating system satisfies the specification given by the abstract operating system. Composing these two results gives us the theorem that the operating system implements isolated tasks.

The verification of the operating system is nearly complete. We have specified all three layers (task, abstract operating system, and target machine) in Rose (i.e. the Boyer-Moore logic). The proof that the abstract operating system implements isolated tasks is complete. The proof that the target machine running the machine code operating system implements the abstract operatng system is nearly complete. We have verified a clock interrupt handler, an error trap handler, and the send and receive supervisor services. The input and output services plus the I/O interrupt handlers remain to be verified. Verifying these routines should pose no significant new problems.

3.3 Systems Programming Language

Our systems programming language is "Micro Gypsy", a small subset of Gypsy comparable to Small C which is defined formally in [23, 15]. The compiler for Micro Gypsy will be verified in Rose, providing a verified translation link between the high level language and the assembly language of the target machine. The target language is an abstract assembly language for the FM8501, the microprocessor which has also been verified in Rose.

Micro Gypsy contains a large part of the sequential component of Gypsy, including exception handling. Principal features of Gypsy not included (at present) in the subset are dynamic data structures, concurrency, and data abstraction. Early experience with Micro Gypsy has convinced us that it contains sufficient functionality to code many of the examples in the literature of full Gypsy.

The compiler for Micro Gypsy is being written in Rose (i.e. the Boyer-Moore logic) and proven in the Rose verification system (i.e. the Boyer-Moore theorem prover, with some modifications). Major components of the compiler and its specification include the following.

- A **pre-processor** translates from Gypsy syntax into a LISP-like prefix syntax. In the process it eliminates all expression evaluation in favor of calls to standard Micro Gypsy procedures.
- A **recognizer** checks the output of the pre-processor for acceptability to the translation process. The recognizer will eventually be obviated when it is proven that the pre-processor always generates acceptable input to the Micro Gypsy compiler.
- The **Micro Gypsy Interpreter** provides an operational semantics for Micro Gypsy. Its input is a program in prefix form and a legal Micro Gypsy state; the result is a state.
- The **assembly language interpreter** provides an analogous operational semantics for the target language.
- The **translator** takes as input a legal Micro Gypsy program and produces a semantically equivalent program in the assembly language.
- Several **mapping functions** translate between Micro Gypsy and assembly language states.

The correctness theorem for the compiler states that a Micro Gypsy program interpreted on a legal Micro Gypsy state is semantically equivalent (under the mappings) to its translated version interpreted on the corresponding assembly language state. The formal statement of the theorem and more discussion are given in [4].

The following progress has been made under the current contract.

1. A complete definition of Micro Gypsy was formulated and documented in a draft manual [23]. Additionally, examples of the use of Micro Gypsy were devised to illustrate the translation of Micro Gypsy syntax to the abstract prefix syntax [24, 25].
2. A preprocessor was written; details are given in the next subsection.
3. The two interpreters, recognizer, translator, and mapping functions were each written as Rose functions for the complete subset.
4. The proof of correctness was begun.

The proof strategy which we evolved was to verify the compiler with a minimal subset of the language and successively add features until we obtained the desired functionality. We currently have a proof of a very simple version of the system with only four instructions: NO-OP, SIGNAL, PROG2, and LOOP, and which only allows references to simple variables. This has given us an enhanced respect for the complexity of the task which remains, but also a wealth of insight into the strategies required to complete it. We envision adding the instructions IF, BEGIN-WHEN, and PROC-CALL and adding data structures ARRAY and RECORD.

3.4 Micro Gypsy Parser

A parser for Micro Gypsy was written in Rose. In the context of the previous subsection, this is the *preprocessor* for translating Micro Gypsy programs into a Lisp-like syntax which is recognized by the *recognizer*. The parser converts a string of characters, representing a micro-Gypsy program, into the form expected by the micro-Gypsy compiler. There are five components:

1. The reader converts a character string into a sequence of tokens, e.g., numbers, names, and keywords.

2. The **tree constructor** converts a sequence of tokens into a tree representation of the original Gypsy syntax. This component marks as errors tokens that do not fit into the Gypsy syntax.
3. The **prefix constructor** converts the Gypsy syntax tree into a prefix form that is similar to compiler input and is more convenient for subsequent processing.
4. The **parser** proper checks that the Gypsy tree represents a legitimate micro-Gypsy program, marking errors such as type inconsistencies and undefined names. This component also simplifies some Gypsy constructs. For example, it converts case statements to if statements, removes expressions from actual parameter lists, and simplifies structures that handle exception conditions.
5. The final component flattens the Gypsy namespace structure. It provides a single list of procedure definitions, which are no longer divided into Gypsy scopes. This component also constructs a type table, containing fully expanded definitions of all types in the program.

The Rose parser was modified to run in Lisp. The Lisp version was tested successfully on several micro-Gypsy examples.

There was some progress toward proving that parser output is acceptable to the recognizer for micro-Gypsy compiler input. This work was centered on the acceptability of the type table. Specification functions were written for the part of the parser relevant to type table construction. A paper proof that the type table is acceptable to the recognizer, on the assumption that the parser satisfies its specification, is near completion.

3.5 Computer Security

Computer security certification is a likely immediate beneficiary of our work on Micro Gypsy and Ava because important progress that is now being made in using normal Gypsy software proofs methods to prove computer security [26].

A non-interference model of security has been devised and proved for the Honeywell SAT system abstract model [26]. A non-interference model for the low water mark problem was specified and proved correct both in Gypsy and Boyer-Moore [18]. Each version had advantages and disadvantages and we expect to exploit our observations made in [18] in designing Rose.

4 Proving Functional Systems

Thus far, we have focused primarily on Rose as a logic and a specification language. However, Rose is executable and can, in principle, be used to implement systems. We imagine Rose eventually being used as a functional programming language. The primary attraction is simplicity: both hardware and software systems can be specified, implemented, and proved in a single formalism.

We are expecting the computing world to make great strides in finding efficient implementations of functional languages. Several interesting such developments have already been taking place in the last few years, including specialized hardware for graph reduction [21], the G-machine implementation on conventional hardware [17, 20], and compilation techniques such as the serial combinator approach [16]. The seeming potential for the exploitation of concurrency through functional languages is well recognized, and may cause a breakthrough in performance. However, even now, there are important applications (where efficiency is not so much of an issue) for Rose as a programming language.

4.1 Functions as Systems

We have proved properties of cooperating sequential functions (a simple multiplexor/demultiplexor system), as described in the status report for the first quarter of 1985. Some theorems were also proved about a version of the OSIS flow modulator, whose Gypsy version is described in [10]. During this contract, however, these theorems were proved in Rose (i.e. with the Boyer-Moore theorem prover). The Boyer-Moore version of the specification is more abstract than the Gypsy version, in that the input stream is a list of "messages". Theorems about this MFM were also stated that are much stronger than the corresponding (proved) Gypsy statements.

I. ICS Technical Reports since January 1985

TR#	Date	Author	Title (some abbreviated)
59	May 87	Kaufmann/ Young	Comparing Gypsy and the Boyer-Moore Logic for Specifying Secure Systems
58	Apr 87	Shankar	Proof-Checking Metamathematics
57	Apr 87	Kim	On Automatically Generating and Using Examples in a Computational Logic System
56	Apr 87	Kim	Measure Guessing: On Experiment with Hypothesis Generation from Examples
55	Feb 87	Boyer/Moore	User's Manual
54	Feb 87	Bevier, Hunt, and Young	Toward Verified Execution Environments
53	Dec 86	Chou	Methods and Examples in Mechanical Geometry Theorem Proving
52	Nov 86	Boyer/Moore	The Addition of Bound Quantifiers and Partial Functions to the Boyer-Moore Logic and Theorem Prover
51	May 86	Cohen	Proving Gypsy Programs
50	Jul 86	Chou	Proving Geometry Theorems Using Wu's Method
49	Dec 85	Chou	Proving and Discovering Geometry Theorems Using Wu's Method
48	Feb 86	Good/Akers/ Smith	Report on Gypsy 2.05
47	Dec 85	Hunt	FM8501: A Verified Microprocessor
46	Jan 85	Kim	EGS: A Transformational Approach to Automatic Example Generation
45	Jan 85	Shankar	A Mechanical Proof of the Church-Rosser Theorem
44	Jan 85	Boyer/Moore	Integrating Decision Procedures into Heuristic Theorem Provers

II. Internal ICS Notes since January 1985

Note Number	Date	Author	Title (some abbreviated)
237	Feb 87	Akers/ L. Smith	An Introduction to the NQTHM Interpreter
236	Feb 87	Kaufmann	A Mechanically-checked Semi-interactive Proof of Correctness of Gries's Algorithm for Finding the Largest Size of a Square True Submatrix
235,234	Feb 87	Kaufmann	A Primitive User's Manual for an Interactive Version of the Boyer-Moore Theorem-Prover (Parts 1 & 2)
233	DRAFT	Siebert/ Akers	Internal Representation of Micro-Gypsy
232	Nov 86	L. Smith	THM Mode
231	Oct 86	Young	A Queue Package in Micro-Gypsy
230	Oct 86	Akers	A Design for an NQTHM Interpreter
229	Oct 86	Kaufmann	"NQTHM" Version of Boyer-Moore
228	Oct 86	L. Smith	Backup
227	Sep 86	Good	Foundations
226	Sep 86	Akers	Gypsy 2.1 Predefined Function and Statement Decriptions
225	Sep 86	Akers	Justification for the New-GVE Implementation
224	Sep 86	Akers	Justification for the Gypsy 2.05 Dialect
223	Sep 86	Akers	A Proposal for Revising Gypsy Hold Spec Requirements
222	Sep 86	Akers	Internal Representation of Executable Micro Gypsy
221	Aug 86	Good	The Formal Definition of Micro Gypsy
220	Aug 86	Bevier	The Correctness of a Small Operating System
219	Jul 86	Akers	Discussions of GVE Alternation Causes
218	Jun 86	Young	The Semantics of Micro Gypsy
217	Jun 86	Young	Horner's Algorithm in Micro Gypsy
216	Jun 86	Young	A Recognizer for Micro Gypsy
215	May 86	Akers	The White Rose Window Interface
214	May 86	Good	DRAFT-In Support of THM
213	Apr 86	Young	Proofs
212	Apr 86	Young	The Low Water Mark Problem Using Non-Interf.
211	Apr 86	Young	The Factorial Example
210	Jan 86	Bevier/ Cohen	On the Well-Definedness of Gypsy Expressions
209	Jan 86	Akers	Gypsy Data Abstraction
208	Jan 86	L.Smith	Gypsy Dialect
207	Dec 85	Good	Rose Development System
206	Dec 85	Good	The Rose Function Space
205	Dec 85	Good	Bootstrapping Techniques
204	Oct 85	Good	Lisp in Rose
203	Oct 85	Good	Rose 84
202	Jan 86	B.Young	Gypsy Paginator
201	Nov 85	LSmith	Gypsy mode in Zmacs
200	Oct 85	MSmith	Reponses to Gypsy Critiques
199	Oct 85 draft	Akers	Gypsy 2.0 GVE Implementation Variances: 10-Oct-85

198	Sep 85	Akers	The Automated GVE Testbed
197	Oct 85 draft	Good	Proving Computing Systems in Ada
196	Sep 85	Akers	Bug Tracking Procedures
195	Sep 85	Akers	Implementation Proposals for Abstract Equality
194	Sep 85	LSmith	Gypsy Interface with TSV05 Magtape
193	Sep 85	Good/ M.Smith	Software Verification in Gypsy
192	Sep 85	Good/ McHugh	Information Flow Tool for Gypsy
191	Sep 85	Good..	Building Software Economically with Mechanized Logic
190	Aug 85	Cohen	New GVE File Directories
189	Aug 85	L.Smith	Burning Gypsy programs into PROM
188	Sep 85	Good	Notes on Revised SPECIAL and ENHANCED HDM
187	Jul 85	Bevier	The Multics Maclisp Version of the GVE
186	Jun 85 TBD	Young	Security in an Abstract Setting
185	Oct 85	Good	Proof of Ordered Search
184	Jun 85	Good	Gypsy Ordered Search
183	Oct 85	Good	Proof of Linear Search
182	Jun 85	Good	Gypsy Linear Search
181	Sep 85	Good	Proof of Object Array Theory
180	Jun 85	Good	Gypsy Object Array Theory
179	Sep 85	Good	Proof of Ordered Object Theory
178	Jun 85	Good	Gypsy Ordered Object Theory
177	Jun 85	Good	Proof of Two Channel Mover II
176	Jun 85	Good	Gypsy Two Channel Mover II
175	Oct 85	Good	Proof of Two Channel Mover I
174	Jun 85	Good	Gypsy Two Channel Mover I
173	Jun 85	Good	Proof of Carrier Connection
172	Jun 85	Good	Gypsy Carrier Connection
171	Jun 85	Akers	Comparison of FORMAT directives
170	Sep 85	Good	DRAFT Notes on FDM
170A	Sep 85	Good	Notes on FDM
169	Sep 85	Good	Notes on Affirm
168	May 85	Good	Gypsy IO without Buffers
167	Apr 85	Good	Micro Filter: Variation #4
166	Apr 85	Good	Micro Filter: Variation #3
165	Apr 85	Good	Micro Filter: Variation #2
164	Apr 85	Good	Micro Filter: Variation #1
163	Feb 85	Bevier	Symbol Table Proofs
162	Feb 85	Bevier	Saddle Back Search
161	Feb 85 draft	Good..	KAIS FEU Issues
160	Feb 85	Good	RSRE Crypto Controller
159	Jan 85	M.Smith	Low Water Mark: Simple Version
158	Jan 85	M.Smith	Low Water Mark Using Abstract Data Type Logs

References

1. Robert L. Akers. A Design for an NQTHM Intepreter. Internal Note #230, Institute for Computing Science, The University of Texas at Austin.

2. Robert L. Akers & Lawrence M. Smith. The White Rose Window Interface. Internal Note #215, Institute for Computing Science, The University of Texas at Austin.

3. Robert L. Akers & Lawrence M. Smith. An Introduction to the NQTHM Interpreter. Internal Note (to appear), Institute for Computing Science, The University of Texas at Austin.

4. W.R. Bevier, W.A. Hunt, W.D. Young. Toward Verified Execution Environments. Proceedings of the 1987 Symposium on Security and Privacy, IEEE, 1987.

5. Robert S. Boyer and J Strother Moore. The User's Manual for A Computational Logic. ICSCA-CMP-55, Institute for Computing Science, The University of Texas at Austin, March, 1987.

6. R. S. Boyer and J. S. Moore. *A Computational Logic.* Academic Press, New York, 1979.

7. R. S. Boyer and J S. Moore. Metafunctions: Proving Them Correct and Using Them Efficiently as New Proof Procedures. In *The Correctness Problem in Computer Science*, R. S. Boyer and J S. Moore, Eds., Academic Press, London, 1981.

8. R.S. Boyer and J S. Moore. The Addition of Bounded Quantification and Partial Functions to the Boyer-Moore Logic and Theorem Prover. ICSCA-CMP-52, Institute for Computer Science and Computing Applications, The University of Texas at Austin, January, 1987.

9. Norman Delisle and Mayer Schwartz. Neptune: a Hypertext System for CAD Applications. CR-85-50, Computer Research Laboratory, Tektronix Laboratories, Tektronix, Inc., January, 1986.

10. Donald I. Good, Ann E. Siebert, Lawrence M. Smith. Message Flow Modulator - Final Report. ICSCA-CMP-34, Institute for Computing Science, The University of Texas at Austin, December, 1982.

11. Donald I. Good. Notes on Affirm. Internal Note #169, Institute for Computing Science, The University of Texas at Austin.

12. Donald I. Good. Notes on FDM. Internal Note #170-A, Institute for Computing Science, The University of Texas at Austin.

13. Donald I. Good. Notes on Revised SPECIAL and ENHANCED HDM. Internal Note #188, Institute for Computing Science, The University of Texas at Austin.

14. Donald I. Good, Robert S. Boyer, J Strother Moore. Trusted Computing Systems from Mechanized Logic. Institute for Computing Science, The University of Texas at Austin, June, 1986.

15. Donald I. Good. The Formal Definition of Micro Gypsy. Internal Note #221, Institute for Computing Science, The University of Texas at Austin.

16. P. Hudak and B. Goldberg. Serial Combinators: 'Optimal' grains of Parallelism. In *IFIP Conference on Functional Programming Languages and Computer Architecture, Nancy, France, September 1985; Lecture Notes in Computer Science 201*,
Springer-Verlag, Berlin, 1985, pp. 382-399.

17. Thomas Johnsson. "Efficient Compilation of Lazy Evaluation". *SIGPLAN Notices 19* (June 1984), 58-69.

18. Matt Kaufmann, William D. Young. Comparing Specification Paradigms for Secure Systems: Gypsy and the Boyer-Moore Logic. submitted to NBS 10th National Computer Security Conference.

19. Richard Kemmerer. Verification Assessment Study Final Report. In 5 volumes, unpublished.

20. Simon L. Peyton-Jones. *The Implementation of Functional Programming Languages.* Prentice-Hall International, London, 1987.

21. Mark Scheevel. NORMA: A Graph Reduction Processor. 1986 ACM Conference on Lisp and Functional Programming, 1986.

22. William D. Young. A Recognizer for Micro Gypsy. Internal Note #216, Institute for Computing Science, The University of Texas at Austin.

23. William D. Young. The Semantics of Micro Gypsy. Internal Note #218, Institute for Computing Science, The University of Texas at Austin.

24. William D. Young. A Queue Package in Micro-Gypsy. Internal Note #231, Institute for Computing Science, The University of Texas at Austin.

25. William D. Young. Horner's Algorithm in Micro Gypsy. Internal Note #217, Institute for Computing Science, The University of Texas at Austin.

26. William D. Young, J. Thomas Haigh. Extending the Non-Interference Version of MLS for SAT. Symposium on Security and Privacy, April, 1986.